FAST APPROXIMATE CONVEX DECOMPOSITION

A Thesis

by

MUKULIKA GHOSH

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2012

Major Subject: Computer Science

FAST APPROXIMATE CONVEX DECOMPOSITION

A Thesis

by

MUKULIKA GHOSH

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Nancy M. Amato
Committee Members,	Jinxiang Chai
	Ergun Akleman
Head of Department,	Duncan M. Walker

August 2012

Major Subject: Computer Science

ABSTRACT

Fast Approximate Convex Decomposition. (August 2012)
Mukulika Ghosh, B.Tech, National Instrtitue of Technology, Durgapur
Chair of Advisory Committee: Dr. Nancy M. Amato

Approximate convex decomposition (ACD) is a technique that partitions an input object into *approximately convex* components. Decomposition into approximately convex pieces is both more efficient to compute than exact convex decomposition and can also generate a more manageable number of components. It can be used as a basis of divide-and-conquer algorithms for applications such as collision detection, skeleton extraction and mesh generation. In this paper, we propose a new method called *Fast Approximate Convex Decomposition* (FACD) that improves the quality of the decomposition and reduces the cost of computing it for both 2D and 3D models. In particular, we propose a new strategy for evaluating potential cuts that aims to reduce the *relative concavity*, rather than absolute concavity. As shown in our results, this leads to more natural and smaller decompositions that include components for small but important features such as toes or fingers while not decomposing larger components, such as the torso that may have concavities due to surface texture. Second, instead of decomposing a component into two pieces at each step, as in the original ACD, we propose a new strategy that uses a dynamic programming approach to select a set of n_c non-crossing (independent) cuts that can be simultaneously applied to decompose the component into $n_c + 1$ components. This reduces the depth of recursion and, together with a more efficient method for computing the concavity measure, leads to significant gains in efficiency. We provide comparitive results for 2D and 3D models illustrating the improvements obtained by FACD over ACD and we compare with the segmentation methods given in the

Princeton Shape Benchmark.

To My Family and Teachers

ACKNOWLEDGMENTS

I could not have accomplished this work without the help and support of an amazing group of people. They have all had important roles to play in this work and also in my life.

First and foremost, I would like to thank my advisor, Dr. Nancy M. Amato, for all her support and guidance. Dr Amato, you were the best advisor that I could possibly have. Thank you for showing me just how much I could achieve. I have learned a great deal from you.

I would like to thank my committee members, Dr. Jinxiang Chai and Dr. Ergun Akleman, for their cooperation, patience and support. I am truly grateful.

I would also like to thank all the members of the Algorithms and Applications group in the Parasol Lab.

Dr. Jyh-Ming Lien and Yanyan Lu provided invaluable support and advice. They gave me the inspiration to strike out in new directions in terms of research ideas.

I have thoroughly enjoyed working with each and every one of the people named above. Again, I would like to thank everyone for their help, support and guidance.

Last but definitely not the least - I would like to thank my family for all their support. I would not be where I am today without them.

TABLE OF CONTENTS

CHAPTER Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

CHAPTER I

INTRODUCTION

The size and complexity of geometric models is continually increasing due to improved tools and techniques for generating them and the computational resources available to render and process them. As such, there is an increased need for techniques to manipulate them. For geometric objects, *convex decomposition* [3] is an appealing strategy as it decomposes the model into convex components which may be easier to process than the original non-convex model. For example, convex decomposition has application in collision detection [4], where the original model is in collision if and only if at least one of its convex components is in collision, and there exist simple algorithms to test collision between convex objects. Unfortunately, algorithms decomposing a large object into exact convex pieces can be computationally inefficient and generate an unmanageable number of components.

Since sometimes minor concavities of the model in the form of surface texture or noise can be ignored, methods have been proposed that decompose an input model into *approximately* convex pieces that are allowed to have concavities that are within some specified tolerance. The idea is that these approximately convex pieces frequently have similar benefits as their convex counterparts, but the decomposition can be computed more efficiently and can result in significantly fewer components. For example, approximate convex components can be used as the basis of simplified representations of objects such as skeletons or silhouettes [5].

The approximate convex decomposition (ACD) algorithm of Lien and Amato [1, 6] uses a greedy approach to decompose an input object into approximately convex

---

The journal model is *IEEE Transactions on Automatic Control.*

pieces. It finds the maximum concave vertex in the model and computes a "good" cut containing that vertex, which decomposes the model into *two* components. These two components are then recursively decomposed until their maximum concavity does not exceed a user-defined tolerance limit. The strategy results in human-perceivable components more efficiently than exact convex decomposition. Various other decomposition algorithms [7] extend the notion to generate natural-looking decomposition of complex models within reasonable time.

There are a few challenges when using the ACD algorithm. First, the selection of an appropriate tolerance can be difficult and input dependent, requiring *a priori* knowledge of the concavity of the input model. Moreover, using the same fixed tolerance to resolve concavity in the decomposed components, can cause over-segmentation in certain areas of the model. Second, applying a *single* cut at every iteration can result in a large number of iterations. When many of the segmentation boundaries are independent of each other, the ACD decomposition process is computationally inefficient because it involves recomputation of concavity information repeatedly. While there have been several recent efforts proposed to improve the quality of ACD [7, 8, 9] using optimization techniques, these methods focus mostly on 2D polygons and usually ignore efficiency issues.

1. Contribution

In this thesis, we propose a method called *Fast ACD* (FACD) that provides higher quality and better efficiency than the existing 2D and 3D ACD algorithms [1, 6, 7, 8, 9].

FACD improves the quality of the resulting decomposition. In particular, instead of a fixed tolerance for the model, FACD computes a *relative concavity* measure for each cut that quantifies the impact of the cut on the concavity with respect to

its surrounding region in the model. As seen in the results, this tends to produce more natural decompositions, e.g., by producing components for relatively small but important features such as fingers or toes while not decomposing around surface texture or undulation that might have similar absolute concavity.

FACD increases the efficiency of the computation. In particular, instead of decomposing into two components with a single cut containing the most concave vertex at each step as in the original ACD, FACD uses an approach to select multiple independent (non-crossing) cuts which are applied in parallel to decompose the model into multiple components. Additionally, we use an optimized method to compute concavity information that reduces redundant computation. Together, these modifications result in significant efficiency gains.

While the improvements proposed can be easily extended to higher genus model, but the experimental results in this thesis are validated for null-genus models.

Chapter II presents an overview of related work and Chapter III gives basic definitions and concepts. An overview of the algorithm is provided in Chapter IV. Chapter V- VII describe the main steps of the algorithm in more detail. Chapter VIII describes some implementation details. Results are presented in Chapter IX.

CHAPTER II

RELATED WORK

Segmentation is a highly studied problem in the literature, see a recent survey [10]. It is usually viewed as an optimization problem that divides the input model while minimizing or maximizing some given criteria or property.

1. Shape Decomposition

Much recent work focuses on creating visually meaningful decompositions. Various decomposition algorithms are provided to decompose a model in two-dimensions [11, 12, 13, 14]. For example, Juengling and Mitchell [14] formulate decomposition of a polygon as an optimization problem and apply dynamic programming to find the optimal subset of cuts from all possible cuts. The objective function used for optimization favors short cuts that create dihedral angles close to π. Mi and DeCarlo [15] propose to decompose a shape into elliptical regions glued together by hyperbolic patches. Their method defines the idea of *relatability* based on smoothed local symmetries that measure how easily two separate curves can be joined together smoothly and naturally. Thus, reasonable cuts are located at places where relatability increases quickly.

In 3D, most segmentation algorithms [16, 17] use clustering algorithms to define the segmentation boundaries and retain or discard them according to the satisfiability of the criteria by the components. Determining the boundaries of the clusters is influenced by topological constraints such as geodesic distance, curvature or dihedral angles. A simplified representation of the input object, such as a curvilinear skeleton as in [18], is sometimes used to aid the decomposition. The underlying approach is to construct the simplified representation, exploit it to define the segment bound-

aries, and then map these segmentation boundaries back to the original model. An algorithm proposed by Aouada and Krim [19] uses a Reeb graph of the model and determines the segments using Morse theory; [8] is similar but uses a convex graph instead. Pre-processing or user assistance is required to construct the curvilinear skeleton or Reeb graph of the input model. Sometimes, to distinguish noise from structural concavities, simplification of the model is used to determine whether to retain or discard cuts. Mesh boundary refinement [20] is often used to obtain smooth and short cuts for decomposition.

To obtain a segmentation close to structural features of the model, most segmentation techniques use multiple deformed models of the same object as input and define the segmentation as a minimization of the error among the poses [21]. In some cases, a pose-invariant measure [22] and multi-dimensional scaling is used to achieve the desired decomposition. However, sometimes certain concavities are underestimated using such scaling.

2. Convex Decomposition

Researchers in computational geometry have studied methods to create convex decompositions subject to some optimization criteria, such as a minimum number of components [3, 23]. Most of these problems are known to be NP-hard. Approximate convex decomposition (ACD) [1] aims at minimizing concavity along with obtaining balanced partitions with perceivable components. Wan [7] extends [1] to incorporate both concavity and curvature and prevents over segmentation by avoiding cuts inside pockets.

Recently, Liu et al. [8] and Ren et al. [9] have proposed to create fewer and more natural nearly convex shapes. Both methods [8, 9] use mutex pairs to enforce the concavity constraint. Points p_1 and p_2 form a mutex pair if their straight line

connection is not completely inside the given shape. Their focus is on separating all mutex pairs whose concavity-based weights are larger than a user-specified threshold. Liu et al. [8] used linear programming to compute the decomposition with minimum cost, and Ren et al. [9] applied a dynamic subgradient-based branch-and-bound search strategy to achieve a minimum number of cuts.

Most of these extensions [8, 9, 7] of ACD focused on 2D polygons. Three-dimensional models are handled by projecting the model into two-dimensions [8, 9]. The projections are segmented and mapped back on the original object. However, the segmentation boundaries become restricted with respect to the choice of the projection planes and their orientation. FACD avoids these problems by working directly on 3D meshes.

CHAPTER III

PRELIMINARIES

A model, M, in this work is a *polygon* or *polyhedron* that is defined by a set of boundaries in $\mathbb{R}^2$ or $\mathbb{R}^3$, respectively. Each boundary consists of a set of connected *edges* in 2D or *faces* (planar polygons bounded by edges) in 3D, respectively. A surface is denoted by a sequence of edges in 2D or by a set of faces connected along edges in 3D. The boundary of a surface is defined as the end-vertices of the poly-line in 2D and the set of external edges bounding the surface in 3D.

The *convex hull* [24] of a model M, CH_M, is defined as the smallest convex set containing M. It is the intersection of all convex sets containing the model. Hence, the convex hull of a convex model is the model itself. *Concavities* are identified as the notches, ditches or holes in the model. More specifically, they are the subset of the model boundary (possibly empty) that are not on the convex hull boundary.

1. Decomposition and Cuts

Decomposition or segmentation divides an input model into a collection of smaller models that can be combined to yield the input model.

Definition .1 *A* **decomposition** *D of model M is defined as a set of components $\{M_i\}$, such that the union of the M_i is the model M and every pair of M_i is interior disjoint:*

$$D(M) = \left\{ M_i | \cup_i M_i = M \text{and } \forall_{i \neq j} M_i^\circ \cap M_j^\circ = \emptyset \right\} \tag{3.1}$$

where M_i° is the open set of M_i, i.e.,M_i excluding its boundary.

Definition .2 *The* ***cuts*** *$\{C_i\}$ in a decomposition D of a model M, are the maximal boundaries of the components M_i that are not boundaries of M. Note that the*

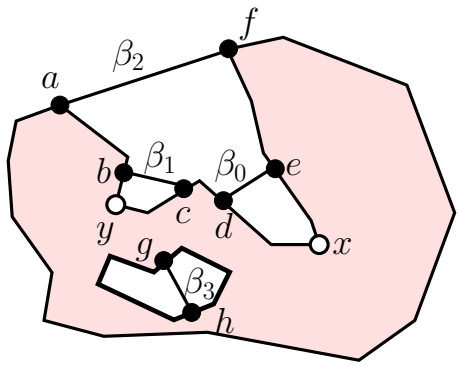


Fig. 1.: Bridges β_0, β_1, β_2, and β_3, and their pocket minima. Bridges β_0 and β_1 are the kids of β_2.

application of $\{C_i\}$ *on* M *gives the components* $\{M_i\}$.

Although in 3D a cut is defined as a set of faces, for convenience we sometimes refer to the cut as the cycle of boundary edges of this set of faces. Hence in this work, a cut denotes a set of faces or a cycle of model edges interchangeably.

2. Bridges and Pockets

A *bridge* is a structure constructed over a concave region in a model. For example, a CH edge or face that is not part of the model can be a bridge or be part of a bridge, as β_2 in Fig. 1 or $\beta 2$ in Fig. 2.

More precisely, a bridge can be defined as:

Definition .3 *A* ***bridge*** β *of a d-dimensional model* M *is a d-dimensional connected surface with no holes such that all vertices in* β *lie on* ∂M*, the boundary of* M*, and the open set of the boundary of* β *is contained in the complement of* M*,* $\overline{M}$.

Therefore in 2D, a bridge is a poly-line that cannot enter polygon M or intersect the boundary of M except at its end vertices, and in 3D, a bridge is a connected polyhedral surface that can intersect M only along the bridge boundary. Note that

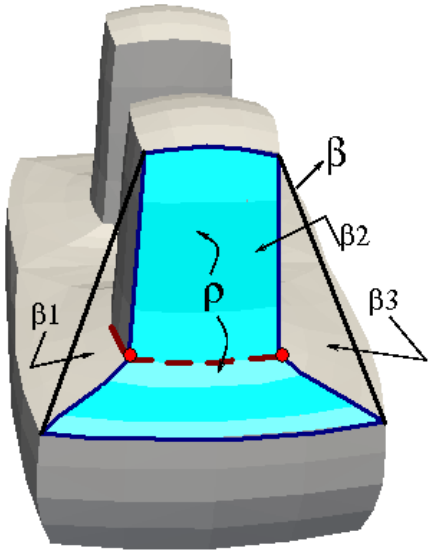


Fig. 2.: Pocket cuts shown in dashed red lines connecting the feature points (red circles) on the bold blue boundaries of the pocket ρ (blue) under the bridge β with faces $\beta 1$, $\beta 2$ and $\beta 3$ that has black boundaries.

this definition of bridge is more general than that in [1] where a bridge was required to be on the convex hull of M (e.g., β_2 in Fig. 1). Examples of bridges in 2D are shown in Fig. 1 and in 3D in Fig. 2.

A *pocket* is the projection or shadow of a bridge on the model surface. Intuitively, it is the region on the model surface below a bridge. It can be defined as follows:

Definition .4 *A* **pocket** ρ *associated with a bridge* β *in a model* M *is a contiguous subset of the surface of* M *that has the same boundary as* β*, so that the region enclosed by* β *and* ρ *is in* $\overline{M}$*, the complement of* M*.*

Intuitively, a bridge can be viewed as a short cut over its pocket while traversing the boundary of M. For example, the pocket of the bridge β_0 in Fig. 1 is a polyline between vertices d and e via x, whereas the pocket of the bridge β in Fig. 2 is the blue region ρ bordered by bold black lines. Note that even though we do not restrict

the bridge to be a convex hull edge or face, the pocket cannot be part of convex hull of the model.

A cut in 2D can intersect with a pocket only at a vertex. In contrast, a cut in 3D can intersect with a pocket in a set of connected edges which is a subset of the boundary cycle of the cut which lies inside the pocket. Each portion of the boundary cycle of a cut that lies in a pocket is referred to as a *pocket-cut*. Hence, pocket-cuts are only defined in 3D.

Definition .5 *A* **pocket-cut** *pc of a cut C in pocket ρ of a 3D model M is a maximal sub-path of the boundary cycle of C that lies entirely in ρ.*

An example of a pocket cut is shown as a dashed line in Fig. 2.

3. Concavity Measurement

Concavities on the surface of the model correspond to pockets, each of which is associated with a bridge. Intuitively, the distance from a vertex in the pocket to its associated bridge provides a measure of its concavity. Other distances such as the distance of the shortest-path from a vertex in the pocket to its bridge can also be defined as a concavity measure for the vertex. Therefore, we define concavity as follows:

Definition .6 *The* **concavity** *of a pocket ρ is the maximum distance from any vertex in ρ to the bridge β of ρ, i.e.,*

$$\text{concavity}(\rho) = \max_{v \in \rho} \text{dist}(v, \beta) \tag{3.2}$$

where dist() *is a distance metric.*

Definition .7 *A* **pocket minimum** *of a pocket ρ is a vertex in ρ realizing the maximum concavity.*

In Fig. 1, x and y are the pocket minima for β_0 and β_1, resp., and x is the pocket minimum for β_2.

Definition .8 *The* **concavity** *of a model M, is the maximum concavity among the pockets in M.*

a. Relative Concavity

All ACD variants [1, 6, 7, 8, 9] define the acceptable tolerance with respect to some absolute measure of the model. Hence, the user must have *a priori* knowledge or an estimate of the concavity of the input structure to obtain a desired segmentation. Also, restricting the tolerance to some absolute measure might overlook certain cavities whose absolute measures are below the tolerance but are of structural importance with respect to their surroundings, e.g.,the toes of the dinopet model in Fig. 3(a). Lowering the threshold of the decomposition based on such concavities results in unecessary decomposition of other regions, e.g., the neck region of the dinopet model in Fig. 3(a).

In this thesis, we introduce a new measure called *relative concavity* to address these issues. Relative concavity quantifies the impact of the cut on the concavity of the model in the surrounding region of the cut.

Definition .9 *The* **relative concavity** *RC of a cut c is the ratio of the concavity of the model M before the application of c to the concavity after decomposition along c. More specifically,*

$$RC(c) = \frac{mc_b}{mc_a} \tag{3.3}$$

where mc_b is concavity(M) *and mc_a is the maximum concavity of M's components after the application of c.*

(a) (b) (c)

Fig. 3.: (a) Approximate Decomposition [1] of dinopet model with tolerance 0.07 to highlight the fingers in hands and feet causing unnecessary segmentation in neck and toe region. No decomposition near tail. (b)(c) Dinopet decomposition using two different thresholds for relative concavity.

Definition .10 *A cut c is said to be* **important** *if its relative concavity is above some user-defined threshold.*

As will be explained later, our new variant of the ACD algorithm computes the relative concavity of the cuts in sub-components of the model which determines the set of cuts that will be applied to decompose the model. As shown in Fig. 3, relative concavity decomposition of the dinopet model enables decomposition around the fingers, toes, and tail without decomposing around the neck.

CHAPTER IV

OVERVIEW

In this thesis, we propose *Fast ACD* (FACD), which is an extension of the ACD algorithm of Lien and Amato [1, 6]. In the original ACD approach, at each recursive application of the method, a single cut that contains a vertex of maximum concavity is used to decompose the current model into two components, both of which are processed recursively. FACD aims to produce a higher quality decomposition more efficiently than the basic ACD for both 2D and 3D models.

To improve the quality of the decomposition, we propose a new measure of concavity that rates cuts based on the relative change they have on concavity (the ratio of the previous to new concavity) rather than on some absolute threshold. This allows the method, for example, to continue to segment the toes on a foot while not decomposing a torso that has some minor surface texture.

To improve efficiency, we modify the algorithm to apply multiple cuts in a single iteration, thus reducing the depths in recursive application. In particular, we propose selecting multiple independent cuts that will decompose the current component into a set of components, each of which meets the required threshold. This is done by first identifying potential cuts. This is the only step in the FACD algorithm which is handled differently in 2D and 3D. The potential cuts are then placed in a weighted "cut-graph" in which edges represent cuts, vertices indicate the components of the model obtained on application of the cuts, and weights are related to relative concavity measurement. Next, dynamic programming is used to select a set of cuts from the cut-graph that will be used to decompose the model. The objective function in the dynamic programming maximizes the total concavity change that maximizes the mutual impact of the cuts over the concavity model.

The basic steps of the algorithm are shown in Algorithm 1. Each step of Algorithm 1 will be detailed in the following chapters.

Algorithm 1 FACD(M,τ)

Input: A model M and tolerance τ.

Output: Components $\{D_i\}$ of decomposed M

1: Find the potential cuts $\{C_k\}$ in M
2: Build a cut-graph G from $\{C_k\}$
3: Use G and τ to select a set of cuts $\{C_r\}$
4: Apply $\{C_r\}$ to decompose M into components $\{D_i\}$
5: **for** each d in $\{D_i\}$ **do**
6: **if** concavity(d) $\geq \tau$ **then**
7: FACD(d,τ)
8: **end if**
9: **end for**

Note that FACD can be exploited to produce hierarchical decomposition: use an initial tolerance to get a canonical decomposition and reduce the tolerance for a sub-component to have a finer decomposition of a particular region corresponding to the sub-component.

CHAPTER V

FINDING INDEPENDENT POTENTIAL CUTS

As stated earlier, the *potential cuts* are the candidates from which multiple cuts are selected to decompose the model in parallel. In this work, the potential cuts and therefore also the final cuts, are required to be independent such that none of them intersect or cross each other except at their boundaries. As described below, this is done differently in 2D and 3D.

1. Potential Cuts in 2D

While cuts in 2D could be poly-lines in this work, we consider cuts that are diagonals of the input model M. In particular, as triangulation of a polygon generates independent diagonals, FACD generates potential cuts using Constrained Delaunay triangulation of a simplified representation of the input polygon. As described in more detail below, we obtain a simplified representation of the polygon by replacing all the vertices in a pocket by a pocket minimum. A set of potential cuts is then obtained by triangulating this simplified polygon [25].

a. Construction of Bridges and Pockets

Convolution (or Minkowski sum) of the input polygon with an α-circle (where α is the diameter of the circle) is used to construct the bridges. The convolution of polygon M with an α-circle includes edges of M translated by $\alpha/2$ in their outward normal direction and arcs with radius $\alpha/2$ centered at vertices of M connecting the end points of the translated edges. As the boundary element can either be a line segment from an edge of M or an arc centered at a vertex of M, an intersection x of two non-adjacent boundary elements of the convolution can be associated with a

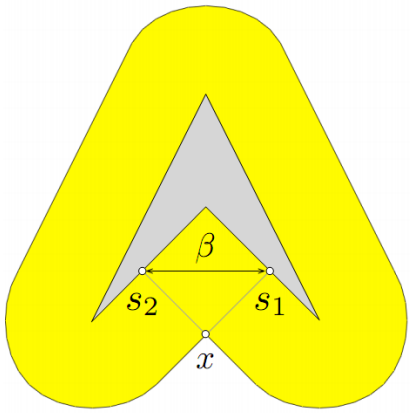


Fig. 4.: Convolution of a V-shaped polygon and a circle. Source pair (S_1, S_2) of an intersection x of the convolution forms a bridge β.

pair of elements of M (i.e., edge or vertex). These edges or vertices are known as the *source pair* of x. For example, in Fig. 4, s_1 and s_2 are the source pair of the intersection point x.

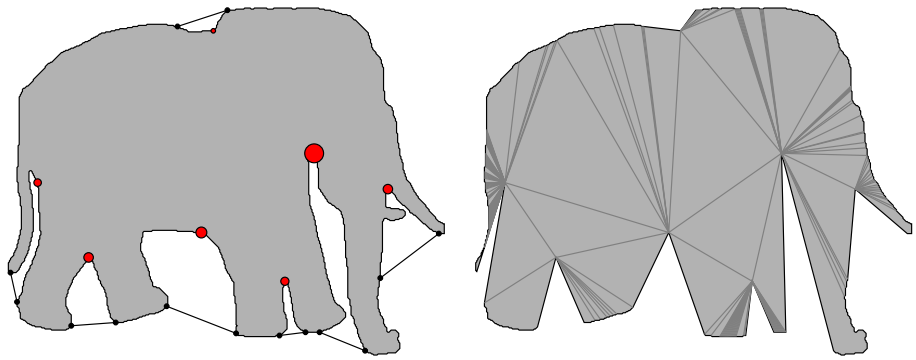
A bridge can be constructed as a segment connecting a source pair if the the segment does not intersect the boundary of M except at the end-vertices of the segment. Hence, if the α-circle at x is empty, then we can create a bridge by connecting the source-pair of x. Thus, a bridge is formed between the tangent points of an empty α-circle and M, which are the source pair of the convolution intersection x. For example, in Fig. 4, the bridge β is created by joining s_1 and s_2 which are the source-pair of x.

To find all bridges for a given α, we have to check whether the α-circles are empty at all intersections. Fortunately, it can be shown that if a single α-circle is empty, then all α-circles from the same orientable loop of the convolution are also empty [26]. A loop is defined to be orientable when the normal directions of the edges in the loop point either all inward or all outward. Hence, to construct the

bridges only a single intersection check is performed per orientable loop.

b. Potential Cuts

Diagonals of a triangulation are non-crossing, i.e., they are independent, and hence are convenient to produce potential cuts. Since Delaunay Triangulations tend to avoid skinny components, the diagonals of a Constrained Delaunay Triangulation (CDT) have been used as cuts by Juenling and Mitchell [14] to create a natural looking decomposition. In FACD, given a polygon M, the potential cuts of M are the diagonals of the CDT of a simplified polygon $\tilde{M}$ of M. As pocket minima realize the concavity of a pocket, the simplified polygon $\tilde{M}$ is composed of the pocket minima and the vertices between every two consecutive bridges. Essentially, $\tilde{M}$ is M with all pocket vertices replaced by the pocket minima. Fig. 5(b) shows a simplified elephant polygon and its potential cuts.



(a) (b)

Fig. 5.: (a) Bridges and pocket minima of the input elephant polygon. The (red) circles are detected pocket minima and the size of the circle indicates the relative significant of the features. (b) The simplified polygon and its candidate cuts.

2. Potential Cuts in 3D

In 3D, construction of potential cuts is extended from the method used in ACD. Construction of cuts in ACD consist of following steps:(1) Identification of pockets by projecting convex hull faces to the model surface; (2) simplification (replacement) of pocket boundaries with feature vertices; (3) creation of pocket-cuts by connecting feature vertices on the pocket boundaries; (4) identification of a cycle of pocket-cuts containing the maximum concave vertex of the model. This cycle is the boundary cycle of the cut used to decompose the model into two components.

FACD also uses Steps(1)-(3) to generate a set of pocket cuts on the model surface. These pocket-cuts are then organized into a pocket-cut-graph. Unlike ACD (which extracts a single cut per recursive step), FACD extracts an independent set of potential cuts from the pocket-cut-graph.

The main steps used to find the independent set of potential cuts are shown in Algorithm 2.

Algorithm 2 Potential Cuts 3D(M)

Input: Model M.

Output: Set of potential cuts $\{C_k\}$.

1: Construct the bridges, pockets and compute scores for pocket-cuts $\{PC\}$.
2: Build a weighted graph, $G(V, E)$ from $\{PC\}$.
3: Find the connected components CC in $G(V, E)$.
4: **for** each connected component cc in CC **do**
5: Extract independent cuts $\{C_i\}$ with largest weights
6: $\{C_k\} \leftarrow \{C_k\} \cup \{C_i\}$.
7: **end for**

In the following sections, we discuss the main steps of Algorithm 2 in more

detail.

a. Construction of Bridges and Pockets

In FACD, pockets and pocket-cuts are constructed as it is done in ACD [1, 6]. A pocket is defined by its boundaries which are also the boundaries of its associated bridge. These boundaries are created by projecting the boundary edges of convex hull faces of the model on the model surface. The projection of a boundary edge of a convex hull face can be defined as the shortest path on the model surface connecting the end-vertices of the edge. Hence the subset of the model surface bounded by the projected boundary edges of a convex hull face constitute a pocket and the convex hull face used in projection forms the maximal element of its associated bridge. Each boundary of a pocket includes a set of connected egdes on the model surface which is then simplified using Douglas Peucker line simplification [27] to determine feature points on the boundaries of a pocket. Pocket-cuts are then created by joining the feature points on the boundaries of a pocket with shortest paths inside the pocket. For example, in Fig 2, the feature points for pocket ρ are shown in red circles and the pocket-cuts associated with them are shown by dashed lines.

b. Potential Cuts

Recall that a potential cut can be represented as a set of pocket-cuts connected in a cycle. To facilitate the selection of an independent set of potential cuts, a pocket-cut graph is constructed.

Pocket-Cut-Graph: The vertices in the pocket-cut graph correspond to pocket-cuts and the edges represent the connectivity among the pocket cuts across common features on shared pocket boundaries. In Fig. 6, the pocket-cut graph for the model in (a) is shown in (b). The cuts a, b, c, d and e in Fig. 6(a) correspond to vertices in

Fig. 6(b).

As in ACD [28], pocket-cuts are scored based on their concavity and curvature. In particular, the weight of the pocket cut pc is given in Equation 5.1.

$$w_{pc} = \kappa_{pc} cv_{pc} \tag{5.1}$$

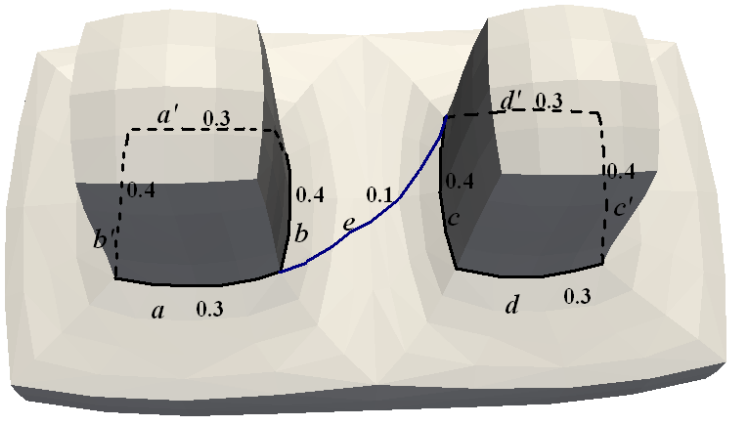
where κ_{pc} is the accumulated curvature [29] of all edges in pc and cv_{pc} is the concavity of pc, which is simply the mean of the concavities of all the vertices of pc.

In the pocket-cut graph, vertices are weighed by the score of their corresponding pocket-cut and edge weights are simply the sum of the weights of their end-points. For example, in Fig. 6(a), let the scores of the pocket-cuts be $a = d = 0.3$, $b = c = 0.4$, and $e = 0.1$. Then, for example, the weight of edge (a, b) in the pocket-cut graph will be $0.3 + 0.4 = 0.7$. See Fig. 6(b).

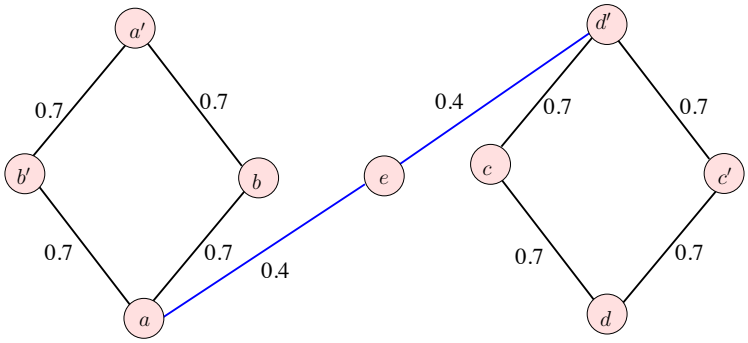
Extraction of Potential Cuts: This step differs in FACD from the original ACD to facilitate handling multiple cuts. A set of vertex-disjoint cycles in the pocket-cut graph corresponds to an independent set of potential cuts. Each connected component of the pocket-cut graph is processed separately to extract independent cycles. Connected components can be classified as: isolated vertex, single cycle, or a general graph. The nature of the connected component determines the method used for extracting the independent cycles from it.

If the connected component of the pocket-cut graph is an isolated vertex, it represents a pocket-cut which is not contained in any cycle. These pocket-cuts might contain important concavities of the model. As shown in Fig. 7, the cut along the neck of the dinopet cannot form a cycle due to the absence of connecting pocket cuts in the neighboring pockets. For such pocket-cut, the end points of the pocket-cut are joined by a shortest path to form a new cycle.

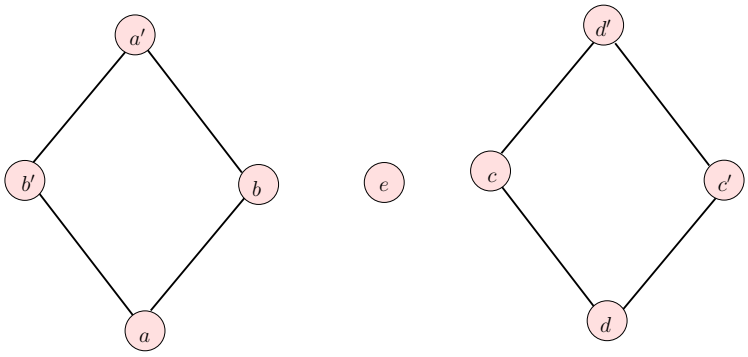
Sometimes, the connected components in the pocket-cut graph are single cycles.



(a)



(b)



(c)

Fig. 6.: Cut graph of a model: (a) Model with high score cuts as bold black and low score as thin blue. (b) Corresponding cut-graph assuming symmetric cuts a',b',c',d' on the opposite side. (c) Cycles extracted from cut-graph.

Such components can be identified when the degree of every vertex in the connected component is 2. Otherwise, the connected component contains more than one cycle. For example, as shown in Fig. 6(b), there exist two cycles in the connected component, i.e., cycles $(ab'a'b)$ and $(dcd'c')$. The edge weights of the pocket-cut graph are used to select between intersecting non-independent cycles. In particular, the all-pair vertex distances are calculated using a modified Floyd-Warshall algorithm. Instead of finding the shortest distance, this algorithm maximizes the distance for every pair of vertices. To achieve this, all edges are initialized to a negative value except the edge in the pocket-cut graph.We augment the pocket-cut graph with additional edges connecting a vertex to itself which also has weight initialized to a negative value. Then, the all-pair vertex distance indicates the maximum-scored path connecting the pocket-cuts corresponding to the vertex pair. If the weight between the vertex pair is negative, then the pocket-cuts corresponding to the vertex pair are disconnected. Hence, if the distance of a vertex to itself is not negative, then there exists a path that starts and ends at the pocket-cut corresponding to the vertex. The path is a cycle and the distance calculated serves as the score of the cycle. For example, in the graph in Fig. 6(b), none of the vertices except e has a negative distance to itself. Thus all vertices except vertex e is part of a cycle and two unique cycles are extracted - $(aba'b')$ and $(cdc'd')$ with scores 2.8 as shown in Fig. 6(c). In case cycles have common vertices in their paths, the cycle with higher score is selected.

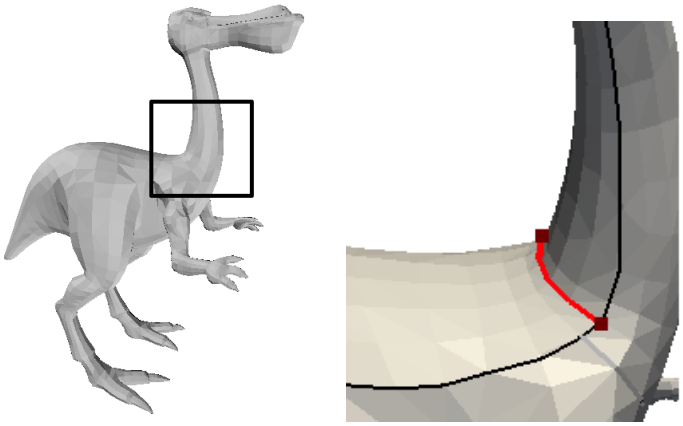


Fig. 7.: Disconnected pocket cuts

CHAPTER VI

CUT-GRAPH

After the independent set of potential cuts is identified, by Constrained Delaunay Triangulation in 2D or by processing the pocket-cut graph in 3D, they are placed in a *cut-graph* to facilitate selection of the final cuts that will be applied to create the decomposition. If all the potential cuts are applied, they will subdivide the model into a set of *primary components*. In the cut-graph, the vertices correspond to these primary components and edges correspond to the potential cuts separating them.

Fig. 8 shows a simple example of cut-graph construction for a two dimensional model. The components 1-18 in 8(a) are vertices in 8(b). The cuts c_1-c_{17} are the edges for the graph in 8(b).

For a null genus model, the cut-graph is a tree. The cut-graphs for higher genus models contain cycles.

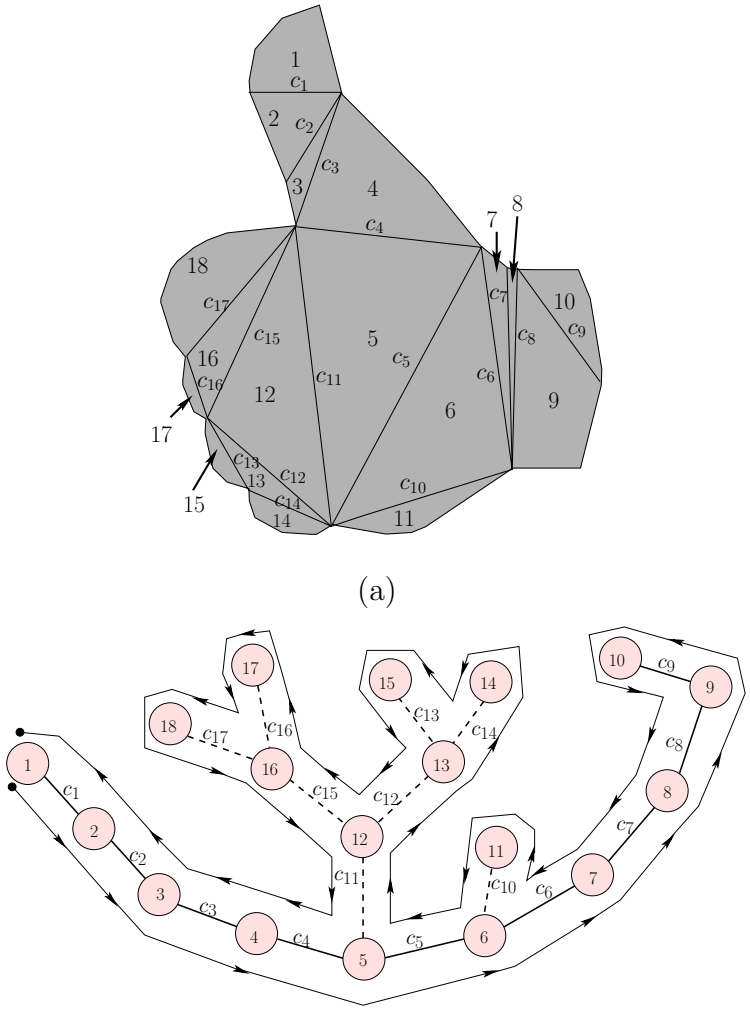


Fig. 8.: Cut-graph example in 2D: (a)The model with the candidate cuts (b) Corresponding cut-graph with bold line showing the backbone and the dotted edges represent branches. Euler tour used for arrangement is shown by arrowed lines covering the graph.

CHAPTER VII

SELECTING FINAL CUTS USING RELATIVE CONCAVITY

With the cut-graph representation, a decomposition can be viewed as a selection of set of edges in the graph. Removal of those edges from the cut-graph partitions the graph into a set of connected components or clusters which represent the decomposed parts of the model. In ACD, a greedy approach for selecting cuts is used. In this work, we apply a dynamic programming approach which attempts to improve the quality and efficiency of the decomposition.

1. Determining a Linear Ordering of Components

A linear ordering of the initial sub-problems is important when applying dynamic programming, e.g., as in matrix multiplication [30]. As the cut-graph encodes the adjacency of the components, an ordering can be specified by a traversal of it. An Euler Tour [31] of an undirected graph corresponds to a depth-first traversal of the graph. In this work, we use an Euler Tour but direct it first along a path realizing the diameter of the graph (the diameter is the longest path in the graph). Then, the rest of the graph can be considered to be branches adjacent to the diameter path. These branches are recursively processed in a similar manner by first traversing the longest path in the branch.

For the graph in Fig. 8(b), the diameter is shown in bold lines, namely the path from 1 to 10 and the branches as the dotted lines. For a vertex containing multiple branches, the largest branch has higher priority over others. For example, in the cut-graph shown in Fig. 8(b), the tour is shown by the arrowed lines surrounding the graph. Hence the arrangement used: 1-2-3-4-5-6-7-8-9-10-9-8-7-6-11-6-5-12-13-14-13-15-13-12-16-17-16-18-16-12-5-4-3-2-1.

For higher genus models, the graph contains cycles. In case of cycles, we separate the branches which form the cycles from the cut-graph and the dynamic programming is applied on the resulting sub-graph. The results from all sub-graphs in the cut-graph are merged to get the final set of cuts. Sometimes, the diameter of the graph might form a cycle (eg., a torus). Such cases are handled by picking the highest weighted edge in the diameter to disconnect the circular path. The edge or the cut selected to break the cycle is considered important and is included in the set of final cuts.

2. Dynamic Programming

Given a set of potential cuts, the final cuts are selected using dynamic programming. The goal is to select the cuts which have maximum impact on the concavity and maximizes the relative concavity measure of important splitting cuts. The primary components, represented as vertices in the cut-graph, are the initial sub-problems of the dynamic programming. Each sub-problem is a subsequence of the linearization of the components obtained from the Euler Tour of the cut-graph. Hence, each sub-problem represents a consecutive region in the input model bounded by some potential cuts.

The objective function of the dynamic programming maximizes the relative concavity measure of important splitting cuts, i.e., those whose relative concavity is above a user-defined threshold. Given a sequence of n components, $S[i, j]$ denotes the sub-problem including component i to component j in the sequence. Let c_k be the splitting cut in $S[i, j]$ joining sub-problems $S[i, k]$ and $S[k + 1, j]$. Then the objective function can be formulated as follows:

$$S[i, j] = \max_{i \leq k < j} \{S[i, k] + S[k + 1, j] + RC(c_k)\} \tag{7.1}$$

where $i < j$ and $S[i, i] = 0$. $RC(c_k)$ denotes the relative concavity measure of the cut c_k. The final cuts in the subproblem $S[i, j]$ include the cuts in sub-problems $S[i, k_{max}]$ and $S[k_{max} + 1, j]$ and the cut $c_{k_{max}}$.

CHAPTER VIII

OPTIMIZATION STRATEGIES FOR EFFICIENCY

To determine the relative concavity for a cut, it is necessary to measure the concavity before and after application of the cut. Similar to ACD [1, 6], we measure concavity as the perpendicular distance from a vertex to hull surface. However, in ACD, the convex hulls used for measuring concavity were built from scratch after each decomposition even when only a small subset of the convex hull was changed by the decomposition. In this section, we describe a method that reuses the existing convex hulls of the sub-components to construct the convex hull of the merged component. During the process of constructing the convex hull of the merged component, the concavity information is also updated simultaneously.

We propose an incremental algorithm to construct the convex hull of merged object from the existing convex hulls of the sub-components. It is assumed that the sub-components to be merged intersect only along a common cut. Hence, the hulls of the sub-components are merged along this common cut. This is done by including or tracing the cut in the hull structures and joining the hulls along the traced cut. However, merging the hulls along the cut might result in a concave structure, especially faces or edges neighboring the cut. Hence a convex envelope is constructed over the cut by iteratively merging pair of adjacent edges (in 2D) or faces (in 3D) starting from those along the cut. The merging is continued with the newly added faces or edges and their neighbors till the resulting structure is convex.

The basic steps of the algorithm are stated in Algorithm 3, and Fig. 9 shows a 2D example of the proposed method.

Algorithm 3 Concavity Estimate(L,R,C)

Input: Hulls of the models to be joined, L and R, and the orientable cut C.

Output: An estimate of the hull of the joined model J with estimate of the concavity for most of its faces.

1: Find the cut C on hull surface of L and R.
2: Merge L and R along C.
3: Populate queue Q with faces adjacent to C.
▷ Q stores possible candidates realizing concavity of the merged structure.
4: **while** Q is not empty **do**
5: $(l, r) = \text{pop}(Q)$.
▷ Iterative merging along the intersection of the hulls till the result structure in convex.
6: **if** (l, r) intersect concavely **then**
7: $J \leftarrow$ Merge (l,r).
8: Update concavity of faces in J.
9: For each face j in J and its neighbor n_j, push(Q,(j, n_j)).
10: **end if**
11: **end while**

1. Merging of Convex Hulls

In 2D, given a diagonal or cut, we first position the cut in the input convex hulls. For example in Fig. 9(a), the cut u,v is shown as a red line segment on the boundary of the input models shown in gray. The convex hulls of the models are shown in bold lines around the polygons. As stated earlier, convex hull edges are also bridges overhead to the concave vertices of the polygon. Sometimes both or one of the cut-vertices might lie inside the input convex hull. In such cases, we identify and

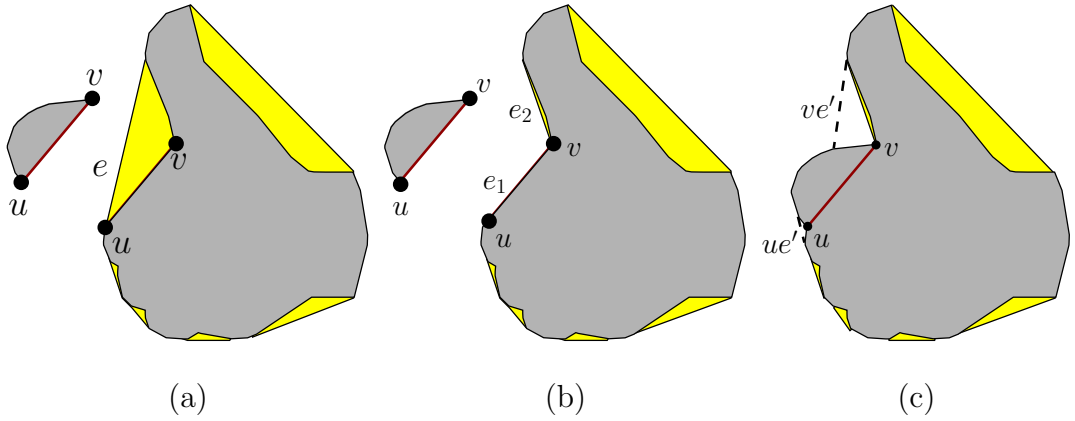


Fig. 9.: Merging convex hulls in 2D: (a) The polygons (shown in gray) with their convex hull (shown in bold lines surrounding the polygon). (b) Subdivision of hull-edge to include cut in the input hull boundary. (c) Joining the convex hulls. Newly constructed hull-edges are shown in dashed lines.

subdivide the bridge *overhead* to the cut-vertex which lies inside the hull such that there exists exactly two incident edges in the hull for each cut-vertex. For the hull in Fig. 9(b), the vertex v lies inside the hull. The hull-edge e overhead of v is thus sub-divided into two hull-edges $e1$ and $e2$.

We then join or merge both the input hulls along the cut-vertices. The resulting structure after joining the hulls in Fig 9(b) along vertices u and v is shown in Fig 9(c). Now, the merged structure might still be concave and in particular, the potential area of concavity will lie along the common cut where the hulls were merged. If an adjacent pair of edges along the traced cut in the merged structure exhibit concavity, then they are replaced by their convex covering (a new edge joining the ends of the adjacent edges). In Fig 9(c), the hull-edges adjacent to both the vertices u,v (shown in bold lines) have concave intersection. The overhead hull-edges ve' and ue' (shown in dashed lines in Fig 9(c)) are thus constructed. The newly created

structure might still be concave along the end-vertices of the newly added edges and hence the process is repeated with the newly added edges and their neighbors till the resulting structure is convex. As shown in Fig. 9(c), the newly added hull-edge ve' shows concave intersection with its neighboring hull-edges and hence iteratively merged to give the final hull of the merged model.

Similarly, in 3D the cut is traced out on both the input hulls. However, subdivision of faces is not as easy as in 2D. It involves projecting the cut-vertices on the input-hulls and splitting the hull faces such that there exists exactly two unique faces in each of the input hulls adjacent for every edge in the cut. As in 2D, the input hulls are merged or joined along the cut. We then search for concavities between pair of adjacent faces along the cut in the merged structure. If two adjacent faces exhibit concavity, they are replaced by their convex hull. This creates new faces which might have concave intersection with their neighboring faces. Hence the process is continued with the newly created faces and their neighbors till no pair of adjacent faces in the resulting structure has concave intersection.

2. Updating Concavity

We now describe how the concavity computation is included with the hull merging process. Note that a hull-edge in 2D is a bridge and a hull-face in 3D can be part of a bridge. Every bridge has a point, known as witness, is associated with its underlying pocket minimmum (a vertex realized maximum concavity). Then, as a new bridge (a edge in 2D or a set of faces in 3D) is constructed, its concavity is computed as given by Equation 8.1

$$c_{new} = \max\{(c_l + d_l), (c_r + d_r), d_c\} \tag{8.1}$$

where c_{new} is the estimated concavity for the newly created bridge, c_l and c_r are the concavities of the witnesses of the bridges merged, and d_l, d_r and d_c are the distances of the witnesses of the former bridges, and of the joining vertex (or vertices), to the new bridge respectively. The witness of the newly created bridge is the projection of the vertex realizing the maximum concavity as computed by Equation 8.1.

CHAPTER IX

RESULTS

All the experiments are conducted on a PC with Pentium CPU and 2GB RAM. Examples of 2D FACD are shown in Fig. 10 and examples of 3D FACD are shown in Fig. 3 in page 12, Fig. 11 in page 35 and Fig. 12 in page 36.

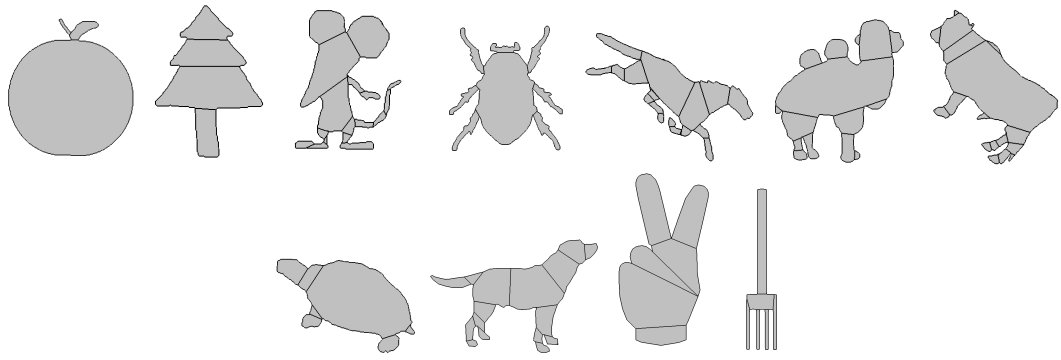
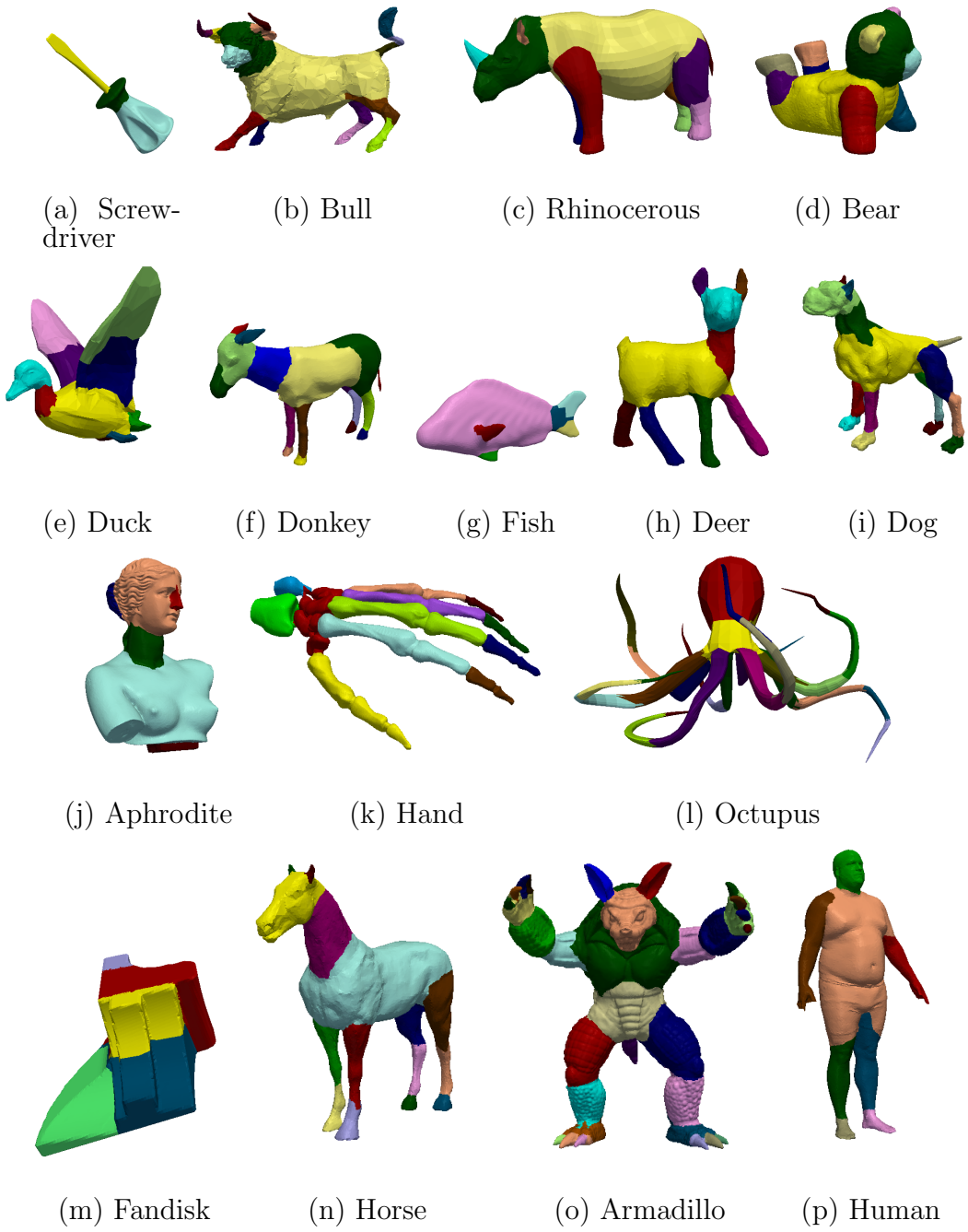


Fig. 10.: 2D FACD using relative concavity threshold

In Fig. 10, structural features such as the leg and hand of the rat model and the leg of the horse model are emphasized. Local concavities or irregularities in the model boundary as in the neck of horse model are not segmented. Similarly in Fig. 11, features important with respect to the structure of the model (for example the fingers in the Armadillo and the ears in the Horse model) are decomposed. However, as the concavity of the model depends on the pose of the input model, certain structural features might be overlooked. As shown in Fig. 11, the horse model has three cuts along the hind legs whereas the fore legs are segmented into two components. The decomposition also depends on the set of potential cuts. As in the Aphrodite model in Fig. 11, no segmentation is obtained along the concavities around the hand as the algorithm considers them as local concavities. Thus, FACD ignores small undulation or irregularities in form of surface noise.



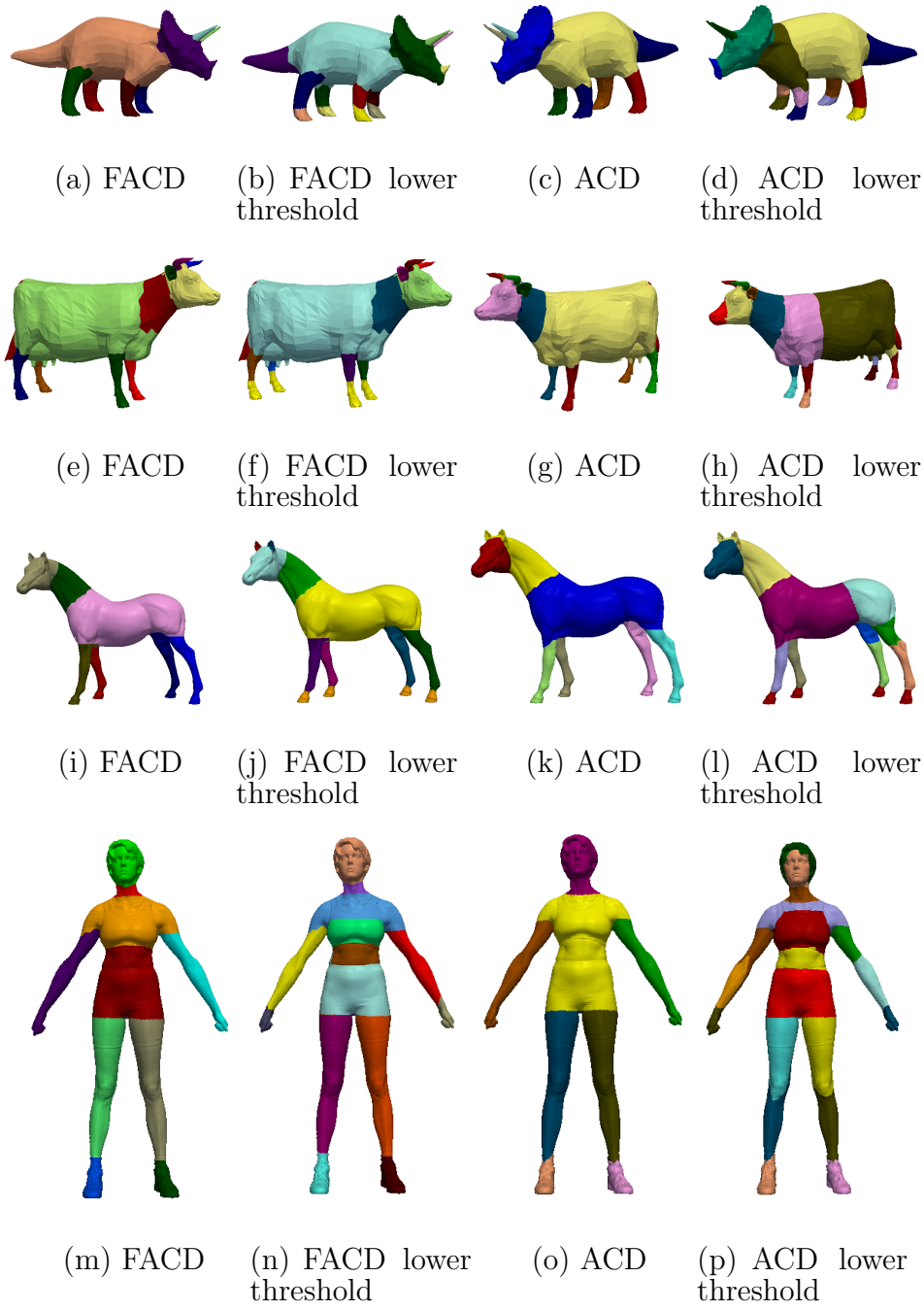
(a) Screw-driver (b) Bull (c) Rhinocerous (d) Bear

(e) Duck (f) Donkey (g) Fish (h) Deer (i) Dog

(j) Aphrodite (k) Hand (l) Octupus

(m) Fandisk (n) Horse (o) Armadillo (p) Human

Fig. 11.: FACD of various models highlighting decomposition along structural features.



(a) FACD (b) FACD lower threshold (c) ACD (d) ACD lower threshold

(e) FACD (f) FACD lower threshold (g) ACD (h) ACD lower threshold

(i) FACD (j) FACD lower threshold (k) ACD (l) ACD lower threshold

(m) FACD (n) FACD lower threshold (o) ACD (p) ACD lower threshold

Fig. 12.: FACD and ACD of various models using two different levels of threshold.

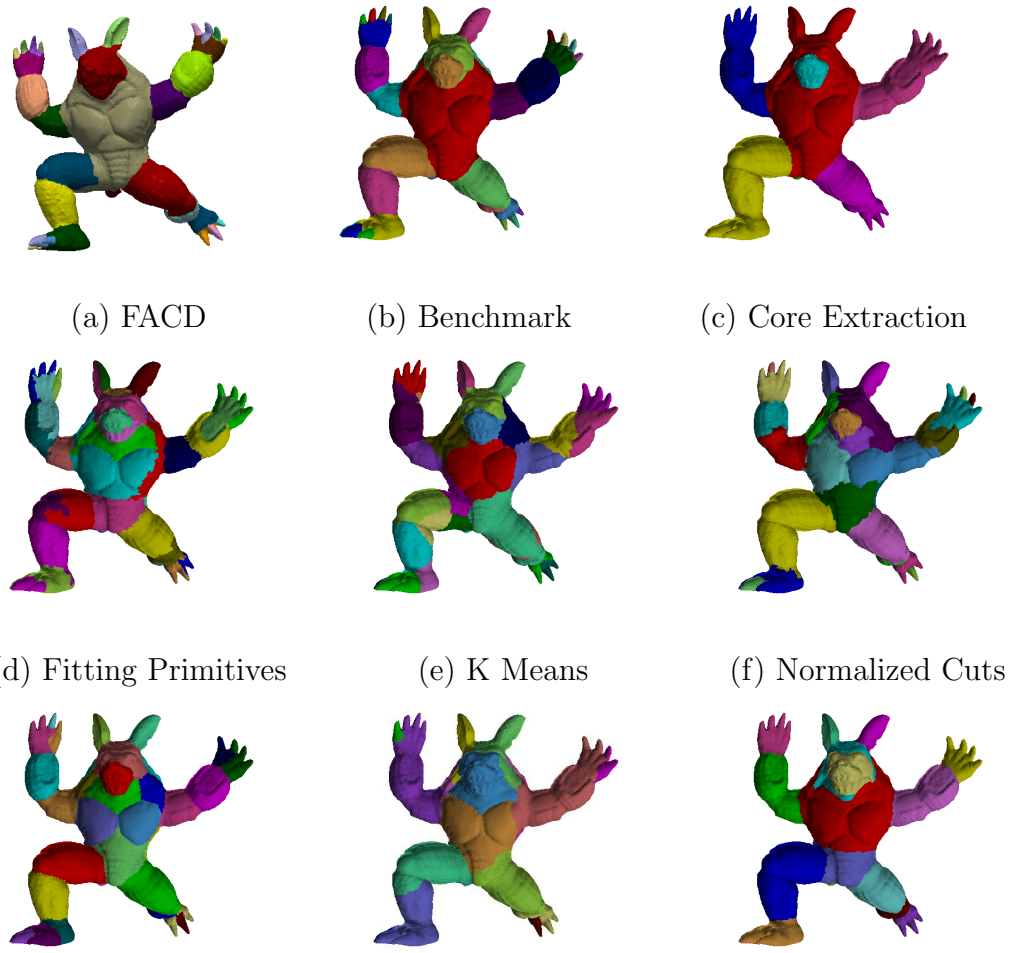
(a) Calf

(b) Bird

(c) Human

Fig. 13.: Hierarchical Decomposition : In decreasing order of tolerance

A hierarchical decomposition can be obtained by decreasing the threshold as shown in Figure 13 in page 37. Fig. 14 and Fig. 15 show a comparison of Armadillo man and a Human model respectively with corresponding human segmentation (Benchmark) and seven other decomposition algorithm available in the Princeton Mesh Segmentation Benchmark [2].



(a) FACD (b) Benchmark (c) Core Extraction

(d) Fitting Primitives (e) K Means (f) Normalized Cuts

(g) Random Cuts (h) Random Walks (i) Shape Diameter

Fig. 14.: Comparison of Model number 281 with the human(benchmark) segmentation and 7 segmentation given in Princeton Benchmark [2]

The comparison result shows that FACD follows closely to the segmentation

(a) FACD

(b) Benchmark

(c) Core Extraction

(d) Fitting Primitives

(e) K Means

(f) Normalized Cuts

(g) Random Cuts

(h) Random Walks

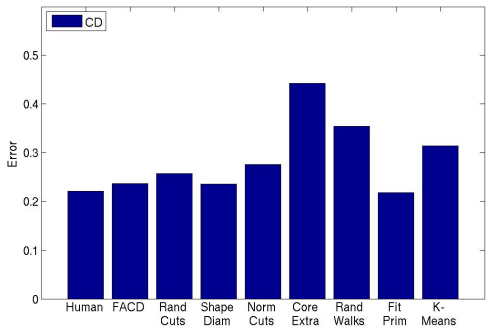
(i) Shape Diameter

Fig. 15.: Comparison of Model number 17 with the human(benchmark) segmentation and 7 segmentation given in Princeton Benchmark [2]

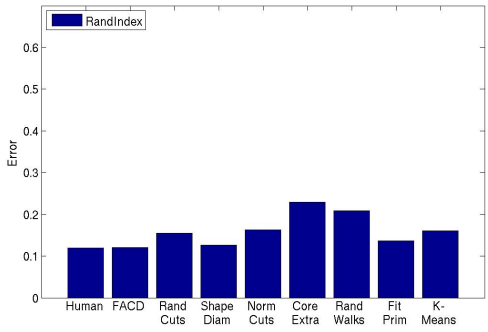
provided by human and hence is more natural looking as compared to the others. The metrics used to evaluate the decompositions in Princeton Benchmark is used to evaluate 5 models from Human and Armadillo category given in the benchmark. The metrics cut discrepancy (distance between segment boundaries), hamming distance (surface area difference), and rand index (likelihood of a pair of faces exist in same segment) are compared in Fig. 16 in page 41. However, the metrics are like shape evaluation metrics and the segmentations compared to (even, human benchmark) depends not only on concavity but other factors like compactness, skeletal structure. For example, over-segmentation might effect in having a low value of cut-discrepancy. Hence the metric varies with experimental models and the constraints used in segmentation method. As shown in Fig. 16, FACD maintains a consistent low value (if not the lowest in all of them) across all metrics and is comparable to the human segmentations.

We next compare FACD results with ACD using concavity improvement, number of components, and time for two different tolerance values. Concavity improvement is specified as the total variance in the concavity of the decomposed parts. It is the difference between the maximum concavity among the decomposed parts and that of original model. Table I in page 42 shows the comparison of the decomposition on the experimental models with respect to the number of components and concavity improvement. However, significant variance in concavity is observed for the female and triceratops model on decreasing tolerance values. This shows FACD decomposes along the areas with significant change in concavity.

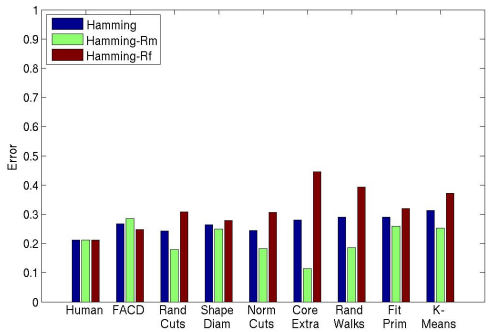
The corresponding decomposition using ACD is also given in Fig. 12 in page 36. Decomposition using a higher threshold in ACD is similar to that of FACD. However, over-segmentation along the torso of the horse and cow model is observed on lowering the threshold in ACD. Using a higher threshold of decomposition hides features



(a) Cut Discrepancy



(b) Rand Index



(c) Hamming Distance

Fig. 16.: Evaluation of metrics using Princeton Benchmark [2]

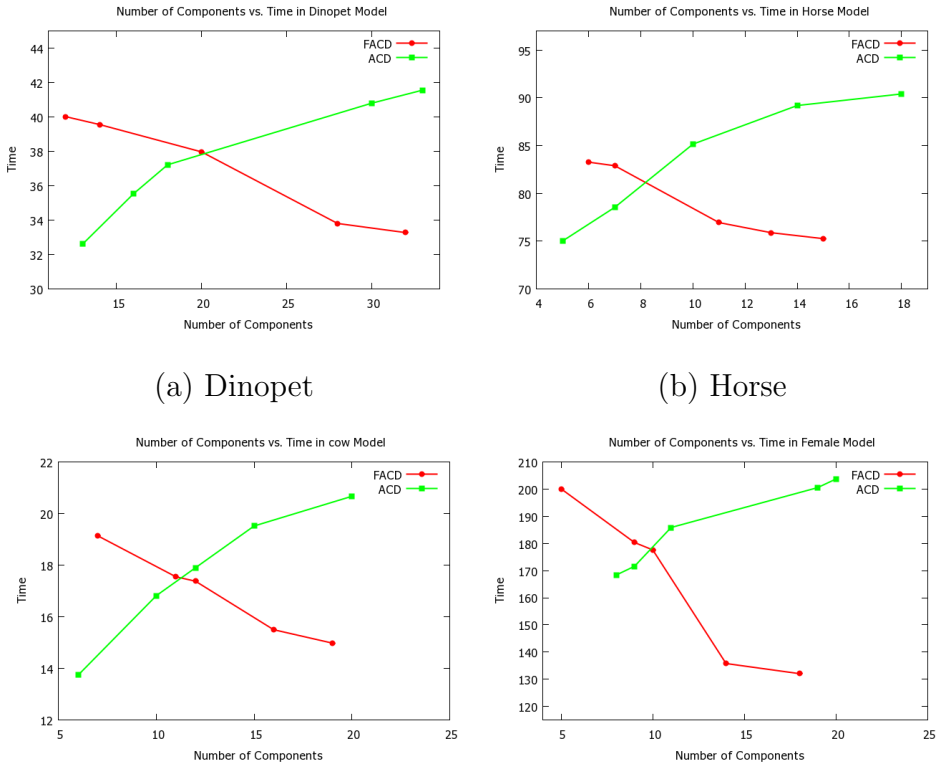
Table I.: Comparison of FACD with respect to (a) Tolerance (τ), (b) number of components, (c) variance in concavity and (d) time.

Models	τ	Size		Variance		Time(in sec)	
		FACD	ACD	FACD	ACD	FACD	ACD
Triceratop	1	8	9	0.398	0.51	16.29	11.08
	0.7	14	15	0.62	0.63	12.45	18.33
Cow	2	12	12	0.526	0.528	17.38	17.9
	0.08	16	20	0.553	0.56	14.5	19.52
Horse	2	7	7	7.54	7.54	82.9	78.55
	0.25	11	14	7.78	8.03	76.95	85.16
Female	2	10	8	13.54	12.31	177.63	168.37
	0.05	14	19	17.28	17.33	135.79	200.56

having small impact on the concavity of the model as ears and hoofs in the Horse model in Fig. 12. However in features such as legs in the triceratops model, certain concavities with lower absolute value of concavity is retained even in higher threshold decomposition without causing decomposition along other regions. Hence, FACD can be used to decompose along low concave features while ignoring noise.

The number of components generated and time taken on application of ACD on the models shown in Fig. 12 in page 36 is given in Table I. As shown in Table I, the time increases with the number of components generated in ACD. As time in FACD is mainly dependent on the number of potential cuts generated, for FACD as in Table I, the time is almost same or decreases with an increase in the number of components. As shown in Table I, the time of the computation does not vary much with the increase in tolerance level. This is because the time of the algorithm

is dependent on the number of potential cuts. In particular, as shown in Figure 17, starting from the same set of potential cuts, the time remains almost the same or even decreases as the threshold decreases and the number of components increases correspondingly. Hence, FACD is more efficient as compared to ACD with respect to higher order of decomposition.



(a) Dinopet

(b) Horse

(c) Cow

(d) Female

Fig. 17.: Time (sec) vs. Number of Components in various models for FACD and ACD

CHAPTER X

CONCLUSION

An approximate convex decomposition algorithm called FACD is proposed which chooses the best cuts for segmentation based on their maximum mutual impact on the concavity of the model. The decomposition is driven by the significance of the potential cuts with respect to their surrounding region as quantified using relative concavity.

Our experimental results show several advantages over the ACD and compares favourably to human segmentation. However, FACD has a few limitations. In three dimensions, our current implementation predicts a set of independent cuts. This restricts the number of cuts in the result. Hence the decomposition of the same input model can be changed by changing the method for prediction of potential cuts.

As the decomposition method uses a bottom-up approach. The method can be extended to aggregation algorithms. The method can also be modified to allow multi-tolerance hierarchical decomposition by varying the threshold parameter for various components or sub-parts of the model.

REFERENCES

[1] J.-M. Lien and N. M. Amato, "Approximate convex decomposition of polyhedra," in *Proc. 2007 ACM Symp. on Solid and Physical modeling (SPM '07)*, 2007, pp. 121–131.

[2] X. Chen, A. Golovinskiy, and T. Funkhouser, "A benchmark for 3D mesh segmentation," *ACM Trans. on Graphics (Proc. SIGGRAPH)*, vol. 28, no. 3, Aug 2009.

[3] B. M. Chazelle, "Convex decompositions of polyhedra," in *Proc. 13th Annual ACM Symp. on Theory of Computing (STOC '81)*, 1981, pp. 70–79.

[4] R. Liu, H. Zhang, and J. Busby, "Convex hull covering of polygonal scenes for accurate collision detection in games," in *Proc. Graphics Interface (GI '08)*, Toronto, Ont., Canada, Canada, 2008, pp. 203–210, Canadian Information Processing Society.

[5] J.-M. Lien, J. Keyser, and N. M. Amato, "Simultaneous shape decomposition and skeletonization," in *Proc. 2006 ACM Symp. on Solid and Physical Modeling (SPM '06)*, 2006, pp. 219–228.

[6] J.-M. Lien and N. M. Amato, "Approximate convex decomposition of polyhedra and its applications," *Computer Aided Geometric Design*, vol. 25, no. 7, pp. 503 – 522, 2008, Solid and Physical Modeling - Selected papers from the Solid and Physical Modeling and Applications Symposium 2007 (SPM 2007), Solid and Physical Modeling and Applications Symposium 2007.

[7] L. Wan, "Parts-based 2d shape decomposition by convex hull," in *Proc. IEEE*

Int. Conf. on Shape Modeling and Applications (SMI 2009), June 2009, pp. 89 –95.

[8] H. Liu, W. Liu, and L.J. Latecki, "Convex shape decomposition," in *Proc. 2010 IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, June 2010, pp. 97 –104.

[9] Z. Ren, J. Yuan, C. Li, and W. Liu, "Minimum near-convex decomposition for robust shape representation," in *Proc. Int. Conf. on Computer Vision (ICCV)*, June 2011.

[10] A. Shamir, "A survey on mesh segmentation techniques," *Computer Graphics Forum*, vol. 27, no. 6, pp. 1539–1556, 2008.

[11] M. Simmons and C. H. Séquin, "2d shape decomposition and the automatic generation of hierarchical representations," *Int. Journal of Shape Modeling*, , no. 4, pp. 63–78, 1998.

[12] M. Tănase and R. C. Veltkamp, "Polygon decomposition based on the straight line skeleton," in *Proc. 19th Conf. on Computational Geometry (SoCG)*. 2003, pp. 58–67, ACM Press.

[13] T. K. Dey, J. Giesen, and S. Goswami, "Shape segmentation and matching with flow discretization," in *Proc. Workshop on Algorithms and Data Structures*, 2003, pp. 25–36.

[14] R. Juengling and M. Mitchell, "Combinatorial shape decomposition," in *Proc. 3rd Int. Conf. on Advances in Vsual Computing - Volume Part II (ISVC'07)*, Berlin, Heidelberg, 2007, pp. 183–192, Springer-Verlag.

[15] X. Mi and D. DeCarlo, "Separating parts from 2d shapes using relatability," in *Proc. IEEE 11th Int. Conf. on Computer Vision, 2007 (ICCV 2007)*. IEEE, 2007, pp. 1–8.

[16] A. Golovinskiy and T. Funkhouser, "Randomized cuts for 3d mesh analysis," in *ACM SIGGRAPH Asia 2008 papers (SIGGRAPH Asia '08)*, Singapore, 2008, pp. 145:1–145:12.

[17] V. Krayevoy and A. Sheffer, "Variational, meaningful shape decomposition," in *ACM SIGGRAPH 2006 Sketches (SIGGRAPH '06)*, Boston, Massachusetts, 2006.

[18] L. Serino, G. S. Baja, and C. Arcelli, "Object decomposition via curvilinear skeleton partition," in *Proc. 20th Int. Conf. on Pattern Recognition (ICPR '10)*, Washington, DC, USA, 2010, pp. 4081–4084, IEEE Computer Society.

[19] D. Aouada and H. Krim, "Meaningful 3d shape partitioning using morse functions," in *Proc. 16th IEEE Int. Conf. on Image Processing (ICIP'09)*, Piscataway, NJ, USA, 2009, pp. 417–420, IEEE Press.

[20] L. Kaplansky and A. Tal, "Mesh segmentation refinement," *Computer Graphics Forum*, vol. 28, no. 7, pp. 1995–2003, 2009.

[21] S. Schaefer and C. Yuksel, "Example-based skeleton extraction," in *Proc. 5th Eurographics Symp. on Geometry Processing*, Aire-la-Ville, Switzerland, Switzerland, 2007, pp. 153–162, Eurographics Association.

[22] S. Katz, G. Leifman, and A. Tal, "Mesh segmentation using feature point and core extraction," *The Visual Computer*, vol. 21, no. 8-10, pp. 649–658, 2005.

[23] B. Chazelle, "A theorem on polygon cutting with applications," in *Proc. 23rd Annu. IEEE Symp. Found. Comput. Sci.*, 1982, pp. 339–349.

[24] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, 2nd ed. Berlin, Germany: Springer-Verlag, 2000.

[25] Y. Lu, J.-M. Lien, M. Ghosh, and N. M. Amato, "Alpha-decomposition of polygons," in *Shape Modeling International*, College Station, Texas, May. 2012.

[26] E. Behar and J.-M. Lien, "Fast and robust 2d minkowski sum using reduced convolution," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS 2011)*, San Francisco, CA, Sep 2011.

[27] D. H. Douglas and T. K. Peucker, *Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature*, pp. 15–28, John Wiley & Sons, Ltd, 2011.

[28] J.-M. Lien, "Approximate convex decomposition and its application," Ph.D. dissertation, Texas A& M University, College Station,US, 2006.

[29] A. Hubeli and M. Gross, "Multiresolution feature extraction for unstructured meshes," in *Proc. Conf. on Visualization '01 (VIS '01)*, Washington, DC, USA, 2001, pp. 287–294, IEEE Computer Society.

[30] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. Cambridge, London: McGraw-Hill Science/Engineering/Math, July 2001.

[31] H. Fleischner, *Eulerian Graphs and Related Topics*, vol. 1 of *Annals of Discrete Mathematics*, Amsterdam: Elsevier Science Limited, North-Holland, 1990.

VITA

Name: Mukulika Ghosh

Address: c/o Dr. Nancy M. Amato
Department of Computer Science and Engineering
3112 TAMU
Texas A&M University
College Station, TX, 77843

Email Address: mghosh@cse.tamu.edu

Education: B.Tech., Computer Science & Engineering,
National Institute of Technology, Durgapur
May 2007

Publications:

1. Yanyan Lu, Jyh-Ming Lien, Mukulika Ghosh, Nancy M. Amato, "Alpha Decomposition of Polygons", *To Appear* Shape Modeling International (SMI), May 2012.